下载APP

27 | 稳定性场景之二：怎样搞定磁盘不足产生的瓶颈问题？

2021-05-21 高楼

高楼的性能工程实战课 进入课程 >

讲述：高楼
时长 11:57 大小 10.95M

你好，我是高楼。

上节课，我们讲解了稳定性场景的两个要点：运行时长和压力量级，并通过课程的示例系统，带你具体操作了稳定性场景。

在定向分析的第一个阶段中，我们分析了虚拟机内存超分导致的操作系统 OOM 的问题，发现是配置的超分过大导致的。在我们降低了虚拟机的内存之后，稳定性场景的运行时间超过了 12 个小时，累积业务量达到 7200 多万，这样的结果已经达到了我们的目标。

可是，由于贪心，我并没有停止场景，就在它继续运行的时候，又出现了新问题……因此，我们今天就进入到定向分析的第二阶段，看看还有什么问题在等着我们。

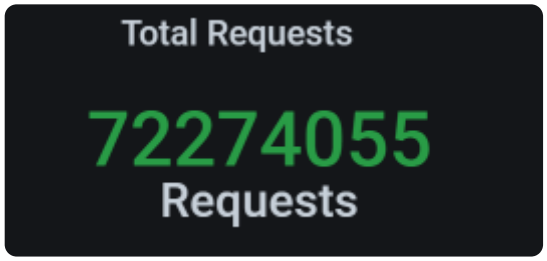
定向监控分析

定向分析第二阶段

当场景继续运行的时候，我看到了这样的数据：

从图中我们可以很明显地看到，在场景持续运行的过程中，TPS 掉下来了，响应时间则是蹭蹭往上涨。

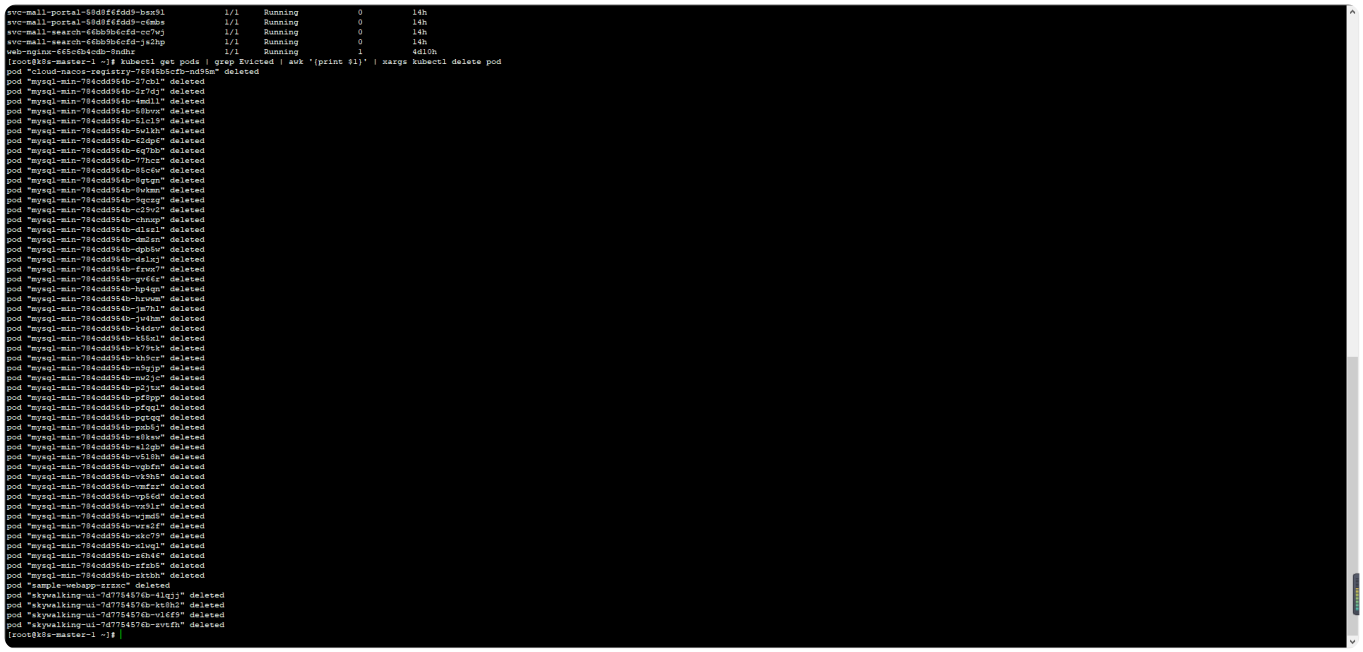
我们看一下这时候的总业务累积量：



也就是说，多了 20 多万的业务累积量。

见到问题，不分析总是觉得不那么舒服，那我们就来分析一下。

还是按照性能分析决策树，我们把计数器一个一个查过去。在我查看 MySQL 的 Pod 日志时，发现它一直在被删掉重建：



请注意，我们这是一个示例系统，为了方便重建，我把 MySQL 放到 Pod 中了。如果是在真实的环境中，我建议你最好根据生产的实际配置来做数据库的配置。

讲到这里，我稍微回应一下行业里的一种声音：数据库不应该放到 Kubernetes 的 Pod 中去。我不清楚持这样观点的人，只是在感觉上认为不安全，还是真正遇到了问题。在我的经验中，很多系统对数据库的性能要求其实并不高，业务量也不大。而用容器来管理便于迁移和重建，并且性能上也完全跟得上。所以，在这种场景下用 Pod 也没有关系。

当然，也有一些系统用数据库比较狠，为了保障更高的性能，会在物理机上直接部署。如果你面对的系统，确实需要用到物理机来创建数据库，那就选择物理机。如果 Pod 可以满足需求，我觉得不用纠结，直接用 Pod 就可以了。

因为 MySQL 的 Pod 被删掉重建，而 MySQL 又位于 worker-1 中，那我们就来看一下 worker-1 的操作系统日志：

```
1 Feb 22 00:43:16 k8s-worker-1 kubelet: I0222 00:43:16.085214    1082 image_gc_m
```

复制代码

原来是分配给 worker-1 的磁盘被用光了，难怪 MySQL 的 Pod 一直在被删掉重建。

我们检查一下磁盘的配额：

复制代码

```
[root@k8s-worker-1 ~]# df -h
文件系统                 容量  已用  可用 已用% 挂载点
devtmpfs                 6.3G     0  6.3G    0% /dev
tmpfs                    6.3G   24M  6.3G    1% /dev/shm
tmpfs                    6.3G   67M  6.3G    2% /run
tmpfs                    6.3G     0  6.3G    0% /sys/fs/cgroup
/dev/mapper/centos-root   47G   45G  2.7G   95% /
/dev/vda1               1014M  304M  711M   30% /boot
tmpfs                    1.3G  4.0K  1.3G    1% /run/user/42
tmpfs                    6.3G   12K  6.3G    1% /var/lib/kubelet/pods/9962f8d2
tmpfs                    6.3G   12K  6.3G    1% /var/lib/kubelet/pods/f5872331
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/e61e5
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/4c057
shm                       64M     0   64M    0% /var/lib/docker/containers/f1e
shm                       64M     0   64M    0% /var/lib/docker/containers/c7e
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/a685a
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/b7da3
tmpfs                    1.3G   36K  1.3G    1% /run/user/0
tmpfs                    6.3G   12K  6.3G    1% /var/lib/kubelet/pods/d01f8686
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/5215b
shm                       64M     0   64M    0% /var/lib/docker/containers/d0c
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/aa512
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/3a7d3
tmpfs                    6.3G   12K  6.3G    1% /var/lib/kubelet/pods/34f60184
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/94000
shm                       64M     0   64M    0% /var/lib/docker/containers/12c
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/36600
tmpfs                    6.3G   12K  6.3G    1% /var/lib/kubelet/pods/251e9c86
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/1defd
shm                       64M     0   64M    0% /var/lib/docker/containers/459
overlay                   47G   45G  2.7G   95% /var/lib/docker/overlay2/49197
[root@k8s-worker-1 ~]#
```

可以看到，磁盘的配置是使用到 95% 就会驱逐 Pod，而现在的磁盘使用量已经到了配置的限额。

既然如此，那我们的解决方案也就非常明确了，就是把磁盘再加大一些，我们再扩大 100G：

复制代码

```
[root@k8s-worker-1 ~]# df -h
文件系统                 容量  已用  可用 已用% 挂载点
devtmpfs                 6.3G     0  6.3G    0% /dev
tmpfs                    6.3G     0  6.3G    0% /dev/shm
tmpfs                    6.3G   19M  6.3G    1% /run
tmpfs                    6.3G     0  6.3G    0% /sys/fs/cgroup
```

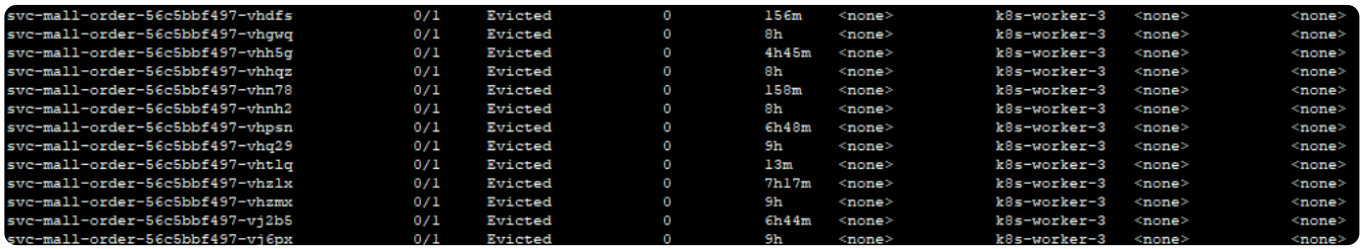
```
/dev/mapper/centos-root  147G   43G  105G   30% /
/dev/vda1               1014M  304M  711M   30% /boot
tmpfs                   1.3G  4.0K  1.3G    1% /run/user/42
tmpfs                   6.3G   12K  6.3G    1% /var/lib/kubelet/pods/d01f8686
tmpfs                   6.3G   12K  6.3G    1% /var/lib/kubelet/pods/9962f8d2
tmpfs                   6.3G   12K  6.3G    1% /var/lib/kubelet/pods/251e9c86
tmpfs                   6.3G   12K  6.3G    1% /var/lib/kubelet/pods/f5872331
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/7380a
shm                      64M     0   64M    0% /var/lib/docker/containers/3c9
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/c841e
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/147a8
shm                      64M     0   64M    0% /var/lib/docker/containers/9e2
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/83e01
shm                      64M     0   64M    0% /var/lib/docker/containers/f23
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/0cfe0
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/7b830
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/0f31c
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/f67a6
shm                      64M     0   64M    0% /var/lib/docker/containers/e3e
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/5cec0
tmpfs                   1.3G   32K  1.3G    1% /run/user/0
tmpfs                   6.3G   12K  6.3G    1% /var/lib/kubelet/pods/704657eb
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/da407
shm                      64M     0   64M    0% /var/lib/docker/containers/76a
tmpfs                   6.3G   12K  6.3G    1% /var/lib/kubelet/pods/a73b0bc6
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/cea9c
shm                      64M     0   64M    0% /var/lib/docker/containers/d93
overlay                 147G   43G  105G   30% /var/lib/docker/overlay2/4c210
[root@k8s-worker-1 ~]#
```

这样就完美地解决了 MySQL 中的 Pod 一直在被删掉重建的问题，对吧！

不过，我们高兴得有些早了，正当我把这次的稳定性场景接着跑起来的时候，结果还没出来，又出现了新问题！于是，我们只得来到定向分析的第三阶段。

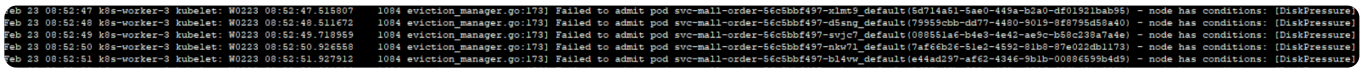
定向分析第三阶段

在上一阶段中，我们发现了 MySQL 的 Pod 被删掉重建。于是，当场景再次运行起来时，我就直接查看了所有 namespace 中的 Pod 状态，结果看到了下面的信息：



不知道什么原因，Order 容器一直在被驱逐。

通过查看日志，我发现了这样的信息：


Feb 23 08:52:47 k8s-worker-3 kubelet: W0223 08:52:47.515807 1084 eviction_manager.go:173] Failed to admit pod svc-mall-order-56c5bbf497-xlmt9_default(5d7f14b1-5ae0-449a-b2a0-df01921bab96) - node has conditions: [DiskPressure]
Feb 23 08:52:48 k8s-worker-3 kubelet: W0223 08:52:48.511672 1084 eviction_manager.go:173] Failed to admit pod svc-mall-order-56c5bbf497-d5snq_default(7995febb-dd77-4480-9015-8f8795d58a40) - node has conditions: [DiskPressure]
Feb 23 08:52:49 k8s-worker-3 kubelet: W0223 08:52:49.718959 1084 eviction_manager.go:173] Failed to admit pod svc-mall-order-56c5bbf497-svjc7_default(088551a6-b4e3-4e42-ae5c-b58c239a7a4e) - node has conditions: [DiskPressure]
Feb 23 08:52:50 k8s-worker-3 kubelet: W0223 08:52:50.926558 1084 eviction_manager.go:173] Failed to admit pod svc-mall-order-56c5bbf497-nkv7l_default(7af66b26-51e2-4592-81b8-87e022db1173) - node has conditions: [DiskPressure]
Feb 23 08:52:51 k8s-worker-3 kubelet: W0223 08:52:51.927912 1084 eviction_manager.go:173] Failed to admit pod svc-mall-order-56c5bbf497-bl4vw_default(e44ad297-af62-4346-9b1b-00886599b4d9) - node has conditions: [DiskPressure]

DiskPressure！这就是说 Order 服务所在的 worker 上的存储被用完了呗。在 Order 应用的虚拟机中，我通过查看 Kubernetes 日志，发现了这样的错误信息：

```
1 kubelet: W0222 14:31:35.507991    1084 eviction_manager.go:344] eviction manag
```

这是 Order 应用的存储已经用完了，并且尝试扩展存储。我们接着查 Kubernetes 日志，看到了“The node was low on resource: ephemeral-storage”的提示。

对 Order 服务来说，不就是写个日志吗？其他的应用方法都是在用 CPU，到底什么 Order 服务会用这么多的磁盘呢？我们查一下日志，结果看到了这样的错误信息：

```
1 [2021-02-25 20:48:22.022] [org.apache.juli.logging.DirectJDKLog] [http-nio-808
2 ### Error querying database.  Cause: java.sql.SQLException: Index oms_order is
3 ### The error may exist in URL [jar:file:/app.jar!/BOOT-INF/lib/mall-mbg-1.0-S
4 ### The error may involve com.dunshan.mall.mapper.OmsOrderMapper.selectByExamp
5 ### The error occurred while setting parameters
6 ### SQL: SELECT count(0) FROM oms_order WHERE (delete_status = ? AND member_id
7 ### Cause: java.sql.SQLException: Index oms_order is corrupted
8 ; uncategorized SQLException; SQL state [HY000]; error code [1712]; Index oms_
9 java.sql.SQLException: Index oms_order is corrupted
```

这样的错误有很多，并且还增加得非常快。难道是 Order 表坏了？

在 Order 服务的日志中，我们可以看到明显是 SQL 执行报错了，但是我们在上面已经增加了 MySQL 机器的硬盘，按理说不应该是 MySQL 的磁盘不够导致的这类报错。所以，我们再来看看 MySQL 错误日志：

```
156 2021-02-25T11:26:35.520327Z 0 [Note] InnoDB: Uncompressed page, stored che
157 InnoDB: Page may be a freshly allocated page
158 2021-02-25T11:26:35.520373Z 0 [Note] InnoDB: It is also possible that your
159 2021-02-25T11:26:35.520408Z 0 [ERROR] InnoDB: Space id and page no stored
160 2021-02-25T11:26:35.520415Z 0 [ERROR] InnoDB: Database page corruption on
161 2021-02-25T11:26:35.520420Z 0 [Note] InnoDB: Page dump in ascii and hex (1
.............
```

从上面的信息来看，应该是数据库的文件损坏了，从而导致了报错。

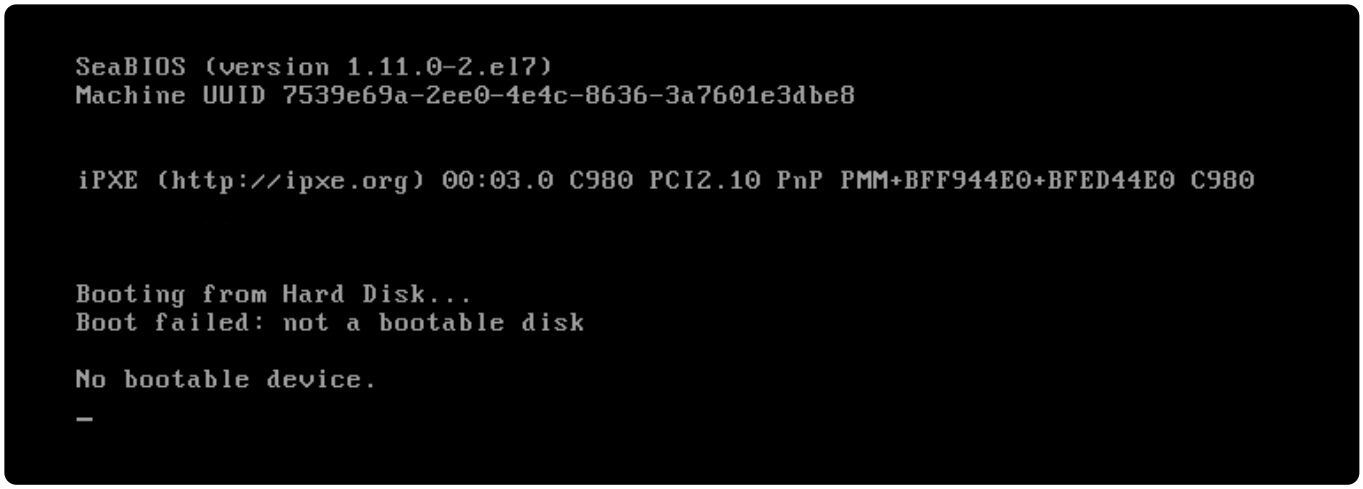
那我们就尝试一下直接在 MySQL 中对这个表进行操作，会不会报错。结果得到如下信息：

复制代码

```
[Err] 1877 - Operation cannot be performed. The table 'mall.oms_order' is miss
```

这个错误日志提示说，操作系统的文件有可能出现了问题，建议重启操作系统。既然系统有建议，那咱就重启一下吧。

然而，世事无常，处处是坑，我们在重启操作系统时看到了这个界面：


SeaBIOS (version 1.11.0-2.el7)
Machine UUID 7539e69a-2ee0-4e4c-8636-3a7601e3dbe8
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+BFF944E0+BFED44E0 C980
Booting from Hard Disk...
Boot failed: not a bootable disk
No bootable device.

想哭对吧？现在居然连磁盘都找不着了……

回想刚才对磁盘的操作，我们在增加磁盘空间时，直接用的是这个命令：

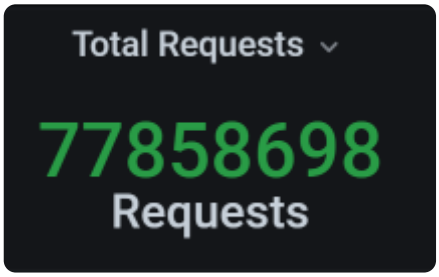
复制代码

```
qumu-img resize imagename +100G
```

我在查看了官方资料之后发现，这个命令虽然能扩容，但是要先关闭虚拟机，不然磁盘会出问题。而我们当时并没有关虚拟机。没办法，我们只有把虚拟机重建一下了。

经过一翻折腾，我们再次把数据库启动，等应用都正常连上之后，再次把稳定性场景运行起来，看到了这样的结果：

此时的业务累积量为：



从场景数据来看，TPS 有所下降，这是因为我们在压力过程中，所有的数据一直都是累加的。累加得越多，数据库中的操作就会越慢，我们看一下 worker-1 中，数据库的资源使用量就可以知道：

你看，这里面的系统负载和 IO 吞吐，都因稳定性运行时间越来越长、数据量越来越多受到了影响。我们后续的优化思路也就非常明确了，那就是保证当前表的数据量级，像分库分表、定期归档这样的手段就可以上了，这些操作都是常规操作。

在我们当前这个场景中，由于稳定性目标已经达到，就不再接着往下整了。毕竟，性能优化是没有止境的。

总结

正如我们前面所说，在稳定性场景中，我们要关注的就是“运行时长”和“业务累积量”这两个指标。只要达到了这两个指标，稳定性场景即可结束。

在这节课中，我们分析了长时间运行导致的两个问题：

磁盘不足的问题

这样的问题在稳定性中也是经常出现的。因此，我们在稳定性场景运行之前，最好预估一下会用到多少磁盘。像日志之类的增长速度，一定要做好循环日志，不要保留太长时间。如果你想保留，那就移到其他地方去。

数据库文件损坏的问题

这是扩展磁盘空间导致的问题。虽然在操作的过程中，磁盘看似成功扩展了，但由于操作的不当导致了数据库文件损坏。这类问题虽然是操作上的问题，但在操作的过程中我们却没有看到任何的错误信息和提醒。

另外，我们还需要充分考虑数据库的累积量。我建议你先通过计算来判断，当数据库的数据量增加后，会对应用产生什么样的影响。磁盘要能保证数据库当前表的数据量，以及数据增长过程中的查询性能。

在这里，我无法穷举出稳定性中的所有问题，只有通过实际案例给你一个分析的思路，供你借鉴。在你的应用场景中，你可以根据 RESAR 性能工程理论分析具体的问题。

总之，在性能工程中，稳定性场景对系统来说是一个严峻的考验。“合情合理地设计稳定性场景，并且做到符合生产业务场景”是我们必须要达到的目标。

课后作业

这就是今天的全部内容，最后给你留两个思考题吧：

1. 你在稳定性场景中遇到过什么由于业务不断累积导致的问题？
2. 在稳定性场景中，如何保证磁盘使用量不会随场景持续增加，而是保持在一个使用量级？

记得在留言区和我讨论、交流你的想法，每一次思考都会让你更进一步。

如果这节课让你有所收获，也欢迎你分享给你的朋友，共同学习进步。我们下一讲再见！

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 26 | 稳定性场景之一：怎样搞定业务积累量产生的瓶颈问题？

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。